CSC 121, Spring 2017 — Large Assignment #1

Worth 10% of the course grade. Due at 1:10pm March 24, to be handed in using MarkUs. This assignment may be handed in late, with a 20% penalty, by 1:10pm on March 28. Assignments will not usually be accepted after that. Contact the instructor as soon as possible if you have a legitimate excuse (such as documented illness) for handing in the assignment late (without penalty).

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you shouldn't leave a discussion with someone else with any written notes (either paper or electronic).

In this assignment, you will write an R function for simulating a queue of people waiting to be served, record the results of multiple simulation runs in a data frame, and produce some plots of the results. The output (text and plots) will be submitted as a single HTML file produced using the `knitr::spin` function. (You will also submit your file of function definitions and your script file.)

You will write an R function called `simulate_queue` that simulates the operation of some establishment, such as a bank, in which customers arrive, may have to wait in line (a queue) until the server (eg, the bank teller) is available, and then have their task done by the server. There is only one server, and only one queue of customers. The establishment opens at what we will call time zero, and closes after a period of time that is specified by an argument of `simulate_queue`. No customers can arrive after the closing time, but all customers already in the queue at closing time are served, regardless of how long this takes.

We will assume that customers arrive independently of each other, and that the probability of one arriving is the same for any time before closing time. The rate at which customers arrive is specified by an argument of `simulate_queue`. The distribution of the time to the next arrival (or the first arrival, after the establishment opens) is given by an exponential distribution with this rate. In R, you can generate a random number from such an exponential distribution with `rexp(1,rate)`. Note that the average time between arrivals is the reciprocal of the arrival rate.

We will assume that the time it takes to serve a customer is independent of the time for other customers (and of other things like arrival times), and that this service time has a uniform distribution over a range whose low and high limits are given as arguments of `simulate_queue`.

The definition of your `simulate_queue` function should start as follows:

```
simulate_queue <- function (open_time, rate, low, high)
```

where the four arguments are as described above (in order).

The `simulate_queue` function should return a list with three elements, named `arrival_times`, `departure_times`, and `queue_lengths`. These three elements will all be numeric vectors of the same length, which is equal to the number of customers served. They contain the arrival times of the customers (which will be between zero and `open_time`, and will be in increasing order), the departure times of these customers (ie, when they are finished being served), and the length of the queue when they arrived (an integer greater than or equal to zero, always zero for the first customer). Note that the number of customers served is not fixed, but will vary from one run to another (if a different random number seed is used).

Here is an example of a call of `simulate_queue` and the value it returns:

```
> simulate_queue(10,0.7,1,2)
$arrival_times
[1] 2.169126 3.339699 3.487038 4.517528 4.926867 6.148282 9.019936

$queue_lengths
[1] 0 1 2 2 3 3 2

$departure_times
[1]  3.995346  5.588048  7.162949  8.498747 10.189931 11.814393 13.164477
```

For this example, I have deliberately not shown what I set the random number seed to before calling `simulate_queue`. Your results will not be the same, since you will use a different random number seed. (Even if you used the same random number seed, they might differ depending on exactly how you wrote your program.) However, if you try several random number seeds, you should see results that are somewhat similar to those above, at least some of the time.

You should also write a function called `run_simulations`, whose definition starts as follows:

```
run_simulations <- function (seeds, open_time, rate, low, high)
```

The `seeds` argument should be a vector of numbers to use as random number seeds. The other arguments correspond to those of `simulate_queue`. This function should call `simulate_queue` once for every element of `seeds`, after setting the random number seed to that element of `seeds` with `set.seed`. The arguments passed to `simulate_queue` should correspond to those passed to `run_simulations`.

The value returned by `run_simulations` should be a data frame with one row for every simulation done, and with columns as follows:

`seed`	random number seed used for this simulation
`open_time`	length of time the establishment is open
`rate`	rate at which customers arrive
`low`	low limit on service time
`high`	high limit on service time
`served`	number of customers served
`maxqueue`	maximum size of the queue when someone arrives
`avewait`	average time customer waits until finished being served
`overtime`	time after closing when last customer finishes being served, zero if last customer served before closing

Note that `open_time`, `rate`, `low`, and `high` will be the same for all simulations done by one call of `run_simulations`. However, you can combine the data frames produced by several calls of `run_simulations` using `rbind`, to get a data frame with results of simulations done with different values for these arguments. The `served`, `maxqueue`, `avewait`, and `overtime` columns will vary from one simulation to the next. They will be computed from the results of each run of `simulate_queue`.

You should put your definitions of `simulate_queue` and `run_simulations` in an R script file called `lga1-defs.R`. If you wish, you may implement these functions using additional functions that you write, whose definitions should also go in this script file.

You should create a separate script file, called `lga1-runs.R`, that runs various simulations, using the functions defined in `lga1-defs.R`. This script file should be run using the `knitr::spin` function, as described in the March 8 lab handout. It should start by setting `knitr` options as specified in the handout, using the `options.r` file from the course web page, or a local copy of this if you prefer. It should then read in your function definitions using `source("lga1-defs.R")`.

You should then show the result of `simulate_queue(10,0.7,1,2)`, which is the example shown above, after setting the random number seed with `set.seed(1)`. Following this, you should show the results of some other simple tests of `simulate_queue` and `run_simulations`, which you devise yourself.

The remainder of the script file should call `run_simulations` four times, to do four groups of fifty simulations. For all groups, `open_time` should be set to 100. The other arguments for `simulate_queue` should be as follows:

Group 1: `rate` = 0.5, `low` = 1.0, `high` = 2.0
Group 2: `rate` = 0.8, `low` = 1.0, `high` = 2.0
Group 3: `rate` = 0.5, `low` = 1.4, `high` = 1.6
Group 4: `rate` = 0.8, `low` = 1.4, `high` = 1.6

For all groups, `run_simulations` should be given a vector of fifty random number seeds, all different (both within and between groups).

You should combine the results of all these simulations into one data frame, using `rbind`, and show the entire contents of this data frame (which will have 200 rows).

You should also produce two scatterplots from these results, with both plots showing the relationship between `served` (on the horizontal axis) and `avewait` (on the vertical axis), with one point shown for each simulation run. The first plot should show the results with `rate` = 0.5; the second plot should show the results with `rate` = 0.8. In both plots, points for simulations in which `low` = 1.0 and `high` = 2.0 should be shown in red, and those for simulations in which `low` = 1.4 and `high` = 1.6 should be shown in green. Both plots should have suitable titles and axis labels.

Your function definitions should be properly indented. The script files of definitions and runs should contain suitable, but not excessive, comments. For the script file run using `knitr`, you can use the special `knitr` format of comments (starting with `#'`) to get nicely formatted text, where this is appropriate. In particular, at the end of the script you should comment briefly on what you might conclude from the plots you show. (Of course, you will not be able to write this comment until you have run an earlier version of the script, and examined the plots.)

You should hand in (using MarkUs) your `lga1-defs.R` and `lga1-runs.R` script files, and the `knitr` output, which will be called `lga1-runs.html`.

Your `simulate_queue` function should be written in what is called an "event-based" manner. This is fairly easy to do for this problem because only two kinds of events occur — a customer may arrive, or a customer may leave after they have been served. Your function can therefore keep track of pending events using two variables, one set to the time of the next arrival, the other to the time when the customer currently being served will be finished (and hence leave). Both of these variables can be set to `Inf` to indicate that no arrival or departure event is currently pending.

The main loop in this function should continue as long there is the possibility of another event. Within this loop, you should see which kind of event will occur next, and update a variable recording the current time to the time of this event. If the next event will be an arrival, you will need to determine when the arrival after that will occur. Also, if the new customer can be served immediately, you will also need to determine when they will finish being served. If the next event will be a departure, you will need to see if another customer is waiting, and if so determine when they will finish being served. You will also need to keep track of the number of customers in the queue, and record the information that `simulate_queue` is supposed to return.

The features we have covered so far are sufficient to do this assignment, but you may also use other features of R that you happen to know, as long as their use does not bypass the point of the assignment. (For example, you may not use an R package that does queuing or other event-based simulations, since that would let you avoid writing a simulation program yourself.)